**REddyProc**

Last updated: 10/29/2014

Current version: 0.6-0

**This R package contains several tools for the processing of half-hourly site-level eddy data. The marginal distribution sampling (MDS) gap filling algorithm and flux partitioning are based on PV-Wave source code from Markus Reichstein**[**1**](#_ENREF_1)**. The ustar filtering is implemented after the ideas of Dario Papale**[**2**](#_ENREF_2)**.**

1 Authors 2

2 Purpose and requirements 2

3 Development steps 3

4 R scripts 5

4.1 Revision control software 5

4.2 Script dependencies 5

4.3 Useful developers’ scripts 5

4.4 Test routines 5

5 Variable formats 6

5.1 Data variable attributes 6

5.2 Data variable naming systematics 6

5.3 Time format and stamp 6

5.4 Abbreviations of data variables newly generated by the gap filling algorithm 7

5.5 Abbreviations of data variables newly generated by the partitioning algorithm 7

6 Description of R5 site specific data frames: 8

6.1 sINFO field of sEddyProc 8

6.2 sDATA field of sEddyProc with input data 8

6.3 sTEMP field of sEddyProc with processing results 9

7 Style guide 10

7.1 Identifiers 10

7.2 Syntax 11

7.3 Diagnostic information 11

7.4 Link requirements 12

8 Documentation 13

8.1 Automated documentation 13

8.2 Automated R5 reference class documentation 14

8.3 Automated example documentation 14

8.4 File naming for automated documentation 14

8.5 Non-automated documentation 14

9 References 14

# Authors

PV-Wave translation, code structure, package development: **Antje M. Moffat (AMM)**

Package maintenance, algorithm evaluation: **Mirco Migliavacca (MM)**

Coding support, online tool implementation: **Thomas Wutzler (TW)**

Plotting routines, user support: **Kerstin Sickel (KS)**

Ustar code: **Olaf Menzer (OM)**

# Purpose and requirements

1. **Online tool**
   1. Format: Automated routines
   2. Code: Stable code with good performance
   3. Data: Input from online submission tool
2. **BGI internal use**
   1. Format: R package format
   2. Code: Re-usable code if applicable
   3. Data: Input from BGI data structure
3. **External (community-wide) use**
   1. Format: R package format (R-Forge)
   2. Code: Encapsulated functionality for easy black box use
   3. Data: Flexible input of user data

# Development steps

The following step are necessary when implementing a new functionality such as ustar filtering into the REddyProc package:

1. Update to the newest version of package development from the repository and re-install package locally
2. Save the script with the new routine(s) in inst/develop
3. Test the code within /inst/develop/testEddyProc.R or the like
4. Write unit tests, e.g. for R5 class methods in test\_sEddyProc.R
5. Regularly submit changes to the repository REddyProc
6. When almost final, move the new script to the main script folder /R
7. Add an example for usage to the official example code sEddyProc.example in R/Eddy.R
8. Generate (automated) documentation and compile package with script /inst/develop/genRpackage.R
9. Install and load the newly generated package archive into R
10. In R, check function documentation and test official example code
11. Implement new functionality into online webtool https://www.bgc-jena.mpg.de/bgi/index.php/Services/REddyProcWeb
12. Document new functionality on webpage
13. When final, submit new package version to R-Forge

# Basic code structure

## R5 reference class

Standard R functions with multiple purposes are provided as standard R functions. Whereas, the functions needed for the processing of the (half-)hourly data are coded in an R5 reference class structure. This has the following reasons (advantages):

Within the sEddyProc class, the dataframes (sDATA and sTEMP) are encapsulated to ensure the correct time stamp and column formatting during the whole processing. On initialization, the internal time stamp format is generated and the columns are checked for numeric type and plausibility with at least three months of continuous data. The two data frames sDATA and sTEMP are global *within* the reference class only.

## Function scheme

General data handling functions:

* **DataFunctions.R, e.g.:**
  + fConvertTimeToPosix()
  + fConvertGapsToNA()

**including test functions, e.g.:**

* + fSetFile()
  + fCheckColNames()
  + fCheckColPlausibility()
* **FileHandling.R, e.g.:**
  + fLoadTXTIntoDataframe()
  + fLoadFluxNCIntoDataframe()
  + fWriteDataframeToFile()
* **GeoFunctions.R, e.g.:**
  + fCalcVPDfromRHandTair()
  + fLloydTaylor()
  + fCalcPotRadiation()

**Standard**

**R functions**

Functions (methods) of **sEddyProc**:

* **Eddy.R, e.g.:**
  + New() – to initialize class
  + sx*Functions*() – for internal use only
  + sExportResults()
* **EddyUstarFilterDP.R:**
  + sEstUstarThreshold()
* **EddyGapfilling.R, e.g.:**
  + sMDSGapFill()
  + sMDSGapFillAfterUstar()
* **EddyPartitioning.R, e.g.:**
  + sMRFluxPartition()
* **EddyPlotting.R, e.g.:**
  + sPlotFingerprint()
  + sPlotHHFluxes()
  + sPlotDiurnalCycle()
  + sPlotDailySums()

**R5 reference class for processing of**

**(half-)hourly eddy data**

## Package script dependencies

DataFunctions.R

GeoFunctions.R

Eddy.R

FileHandling.R

EddyPartitioning.R

EddyPlotting.R

EddyGapfilling.R

EddyUstarFilterDP.R

## Useful developers’ scripts

Developer files and information are in the directory /inst. The subdirectory /develop contains the DevelopmentNotes.docx and scripts. /examples contains different data input and output templates. The R scripts in /scripts are a pure copy of the main scripts in /R so that they get distributed with the package as explicit code. The directory /tests contains unit and integration tests for the different R (sub)routines.

Scripts in /inst/develop:

setREnvir.R (Re-)set user specific R environment

genRpackage.R Generate (update) documenation and package

testEddyProc.R Various test routines to develop and test code

Some of the code that is used during debugging is written inside dummy functions called .tmp.f(). This workaround avoid comments, which are more difficult to execute in an interactive R-session. The workaround .tmp.f() function can be defined several times, and is deleted on loading or sourcing the package code by file zzz.R.

## Test routines

Several unit and integration test routines are implemented using the package **test\_that,** which is more convenient for testing many files in a package and for handling potential errors during testing. All tests provided in the directory /inst/tests are executed when calling the test\_package function of **test\_that**. The file doRUnit.R in the required standard directory /tests simply calls this function during package installation or R-Forge upload.

## Revision control software

Mercurial repository: <BGI>/code/R/Package/REddyProc

# Variable formats

## Data variable attributes

For each data column, the two attributes ‘varnames’ for variable names and ‘units’ for unit names are provided, e.g.

attr(Data.V.n, 'varnames') <- 'Rg'

attr(Data.V.n, 'units') <- 'W\_m-2'

(If not applicable or not provided, the symbol '-' is used as a filler.)

## Data variable naming systematic

* To avoid overwriting of previous results, new columns in sTEMP get unique new names ‘VAR\_’ for renaming prefix to variable name or ‘NEW\_’ for omitting prefix later.
* Suffices are attached with underscore ‘\_’ .
* Further variable information is provided with dot ‘.’, e.g. NEE filtered with the quality flag NEE\_fqc for values of zero is called NEE\_f.NEE\_fqc\_0
* As a dummy variable declaration, the text characters 'none' are used.

## Time format and stamp

For compatibility, the time stamp is converted to POSIX format. This requires the standardization of the **FLUXNET time format**:

* The 24th hour needs to be corrected to 0 o’clock of the next day.
* The 366th day of the year needs to be corrected to the 1st day of the next year (i.e. **1998 1 1 0.0 366** to **1999 1 1 0.0 1** in format 'year month day hour DoY').
* The time zone is set to GMT to avoid daylight saving problems.
* The format is provided in POSIX: POSIXct – calendar time in seconds since 1970, POSIXlt – local time in time structure.

Internal adjustment of the **FLUXNET time stamp**:

* The data is stamped at the end of the measured half-hour. Therefore, the time stamp of one year starts at the second half-hour and ends at the first half hour of the next year (i.e. the range is: 1998-01-01 00:30:00 to 1999-01-01 00:00:00 in POSIX format).
* To attribute the last half hour of the year to the correct year, the half-hours are internally shifted by minus 15 minutes to the***middle***of the measurement period (i.e. the shifted range is: 1998-01-01 00:15:00 to 1998-12-31 23:45:00 in POSIX format).
* When exporting the data, the time stamp is shifted back to the end of the measured half-hour (i.e. the range is again: 1998-01-01 00:30:00 to 1999-01-01 00:00:00 in POSIX format).

## Suffix naming for variables newly generated by the gap filling algorithm

VAR\_orig # Variable with original values of VAR

VAR\_f # Variable with gaps filled

VAR\_fall # Variable with all datapoints filled (for uncertainty

estimates)

VAR\_fnum # Number of datapoints used for filling

VAR\_fsd # Standard deviation of data points used for filling

VAR\_fmeth # Method used for filling

VAR\_fwin # Window size used for filling

VAR\_fqc # Quality flag used for filling

## Variable naming for variables newly generated by the partitioning algorithm

PotRad # Potential radiation

FP\_NEEnight # Good (original) NEE nighttime fluxes used for

flux partitioning

FP\_Temp # Good (original) temperature measurements used for

flux partitioning

E\_0 # Estimated temperature sensitivity

R\_ref # Estimated reference respiration

Reco # Estimated ecosystem respiration

GPP\_f # Estimated gross primary production

# Description of R5 site specific data frames:

sEddyProc <- **setRefClass**("sEddyProc", fields=list(

sID='character' ##<< String with Site ID

,sDATA='data.frame' ##<< Data frame with (fixed) site data

,sINFO='list' ##<< List with site information

,sTEMP='data.frame' ##<< Data frame with (temporary) result data))

## sINFO field of sEddyProc

List of 6

$ DIMS : int 17520 # Number of data rows

$ DTS : num 48 # Number of daily time step (24 or 48 half-hours)

$ Y.START: int 1998 # Starting year

$ Y.END : int 1998 # Ending year

$ Y.NUMS : num 1 # Number of years

$ Y.NAME : chr "1998" # Name for years (e.g. "00-02")

## sDATA field of sEddyProc with input data

'data.frame': 17520 obs. of 5 variables:

$ sDateTime: POSIXct, format: "1998-01-01 00:15:00" "1998-01-01 00:45:00" "1998-01-01 01:15:00" "1998-01-01 01:45:00" ...

$ NEE : num -1.21 1.72 NA NA 2.55 NA NA NA 4.11 NA ...

$ Rg : num 0 0 0 0 0 0 0 0 0 0 ...

$ Tair : num 7.4 7.5 7.1 6.6 6.6 6.5 6.3 6.1 5.9 6.2 ...

$ VPD : num 4.6 4.6 4.3 3.9 3.9 4 3.9 3.7 3.4 3.4 ...

|  | **sDateTime** | **NEE** | **Rg** | **Tair** | **VPD** | **...** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | 1998-01-01 00:15:00 | -1.210 | 0.00 | 7.4 | 4.6 | … |
| **2** | 1998-01-01 00:45:00 | 1.720 | 0.00 | 7.5 | 4.6 | … |
| **3** | 1998-01-01 01:15:00 | NA | 0.00 | 7.1 | 4.3 | … |
| **4** | 1998-01-01 01:45:00 | NA | 0.00 | 6.6 | 3.9 | … |
| **5** | 1998-01-01 02:15:00 | 2.550 | 0.00 | 6.6 | 3.9 | … |
| **6** | 1998-01-01 02:45:00 | NA | 0.00 | 6.5 | 4.0 | … |
| **7** | … | … | … | … | … | … |

## sTEMP field of sEddyProc with processing results

'data.frame': 17520 obs. of 5 variables:

$ sDateTime: POSIXct, format: "1998-01-01 00:15:00" "1998-01-01 00:45:00" "1998-01-01 01:15:00" "1998-01-01 01:45:00" ...

$ NEE\_orig : num -1.21 1.72 NA NA 2.55 NA NA NA 4.11 NA ...

$ NEE\_f : num -1.21 1.72 1.03 1.09 2.55 ...

$ NEE\_fsd : num 2.56 2.53 3.33 3.18 3.16 ...

|  | **sDateTime** | **NEE\_orig** | **NEE\_f** | **NEE\_fsd** | **...** |
| --- | --- | --- | --- | --- | --- |
| **1** | 1998-01-01 00:15:00 | -1.210 | -1.21000000 | 2.55563884 | … |
| **2** | 1998-01-01 00:45:00 | 1.720 | 1.72000000 | 2.53160424 | … |
| **3** | 1998-01-01 01:15:00 | NA | 1.03457895 | 3.33442008 | … |
| **4** | 1998-01-01 01:45:00 | NA | 1.09469697 | 3.17671345 | … |
| **5** | 1998-01-01 02:15:00 | 2.550 | 2.55000000 | 3.16435493 | … |
| **6** | 1998-01-01 02:45:00 | NA | 1.19955224 | 2.90596900 | … |
| **7** | … | … | … | … | … |

# Style guide

The coding guidelines are based on the recommendations of

<http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html>

and BGI internal inlinedocs template

<https://www.bgc-jena.mpg.de/bgi/index.php/Intra/ComputingCodeTemplateFunctionR>.

***Please***follow these guidelines to ensure good readability of the R code and enable its automated documentation*.*

## Identifiers

The variable or function names are composed as ‘***qVariableName.F.t*’**:

* a qualifier ***q*** indicating the type (this also helps the sorting in the workspace view and documentation):
  + ***‘f’*** for function names
  + ***‘s’*** for site specific reference class fields and methods
  + ***‘k’*** for constant variables
  + ***‘g’*** for global variables (should be avoided)
  + *omitted* for all other variables
* variable name with the initial letters of words capitalized, e.g. ***VariableName,*** and, if needed, underscore and unit name, e.g. ***VariableName\_unit***
* followed by the format ***.F*** and type ***.t*** separated by dots:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Variable format** |  | **Variable type** |
| ***.S*** | single value *(optional)* | ***.n*** | numeric |
| ***.V*** | vector | ***.l*** | numeric length |
| ***.M*** | matrix | ***.i*** | numeric index or counter |
| ***.F*** | data frame | ***.h*** | numeric half-hourly vector |
| ***.L*** | (extended) list | ***.d*** | numeric daily vector |
| ***.C*** | class | ***.m*** | numeric monthly vector |
| ***.O*** | overloaded | ***.y*** | numeric yearly vector |
|  |  | ***.s*** | character string |
|  |  | ***.b*** | boolean |
|  |  | ***.f*** | factor |
|  |  | ***.p*** | POSIX time format |
|  |  | ***.x*** | mixed types, e.g. in data frames *(optional)* |
|  |  | ***.o*** | overloaded |

## Syntax

* **Line Length:** The maximum line length is 80 characters.
* **Indentation:** When indenting your code, use two spaces. Never use tabs or mix tabs and spaces.
* **Spacing:** 
  + Place spaces around all binary operators (=, +, -, <-, etc.). *Exception: Spaces around ='s are optional when passing parameters in a function call.*
  + Do not place a space before a comma, but always place one after a comma.
  + Place a space before left parenthesis, except in a function call.
  + Do not place spaces around code in parentheses or square brackets. *Exception: Always place a space after a comma.*
* **Curly Braces:** An opening curly brace should never go on its own line; a closing curly brace should always go on its own line. For consistency use curly braces for single statement blocks.
* **Assignment:** Use <-, not =, for assignment.
* **Semicolons:** Do not use semicolons or put multiple commands in one line.
* **Data Frame Search Path:** Avoid the use of attach().
* **Messages and Errors:** See below under Diagnostic information.

## Diagnostic information

For diagnostic purposes, it is helpful to have one line messages for *each* function operating on or with data. Please use:

* **message()** for diagnostic messages,
* **warning()** for warnings of potential errors,
* **stop()** for fatal errors requiring to stop the execution of the code.

These messages will be printed to screen or log file, respectively. For easier error tracking, the message in subordinate functions should be:

warning/stop(NameOfFunctionCalledFrom.s, ':::ThisFunctionName::: Error!')

In contrast to print() or cat(), output from message() and warning() can be suppressed. The text is formatted similar to paste(sep='').

Example for common formatting of the warnings:

warning('sCallFunction::: Variable (', Var.s, ') contains no data at all!')

## Link requirements

Building of packages requires special handling of loading from R libraries and sourcing of scripts.

* **source()** - There should be no sourcing inside the scripts.
  + All functions within the R directory are loaded on package installation and then the complete code is sourced.
  + Attention: For the generation of the inlinedocs documentation, the script is sourced in the alphabetic order of the file names(!).
* **library()** – There should be no explicit calls to library().
  + The loading of libraries is handled in the DESCRIPTION file. Libraries that are only used by specific functions and therefore do not need to be loaded in general can be handled with:

if( !require(...) ) stop('Required package ... could not be loaded!')

* + They should also be added to the DESCRIPTION file under "Suggests:".

# Documentation

## Automated documentation

The automated documentation is generated with the package **inlinedocs**. Commenting of the code thus needs to comply with the **inlinedocs** rules:

fTemplateInlinedocs<- function(

**##title<<**

## Short function title (mandatory)

**##description<<**

## Description of the function (optional)

Var1.V.n ##<< Description of input variable 1 (mandatory)

,Var2.V.n ##<< Description of input variable 2 (mandatory)

)

**##author<<**

## Name initials of author(s) (mandatory)

**##details<<**

## Long and detailed description of the function (optional)

**##seealso<<**

## List of names of other functions doing similar stuff (optional)

## with links to documentation pages: \code{\link{fTemplateInlinedocs}}

**##references<<**

## List of papers describing the mechanisms or algorithms used (optional)

**# Standard comments (with one hash):** These will not appear in the documentation (optional)

**# TODO(username):** Explicit description of action to be taken (optional)

**# TEST:** Function arguments used for testing only (optional)

{

**##details<<**

## Description of what (this part of) your function does (optional)

plot(Var1.V.n, Var2.V.n)

**##details<<**

## Further description of what (this part of) your function does (optional)

lSum.V.n <- Var1.V.n + Var1.V.n

**##details<<**

## Further descriptions, e.g of lists (for lists use 'describe'):

**##describe<<**

List.L <- list(

item0 ##<< Description of list item 0

,item1 = list( ##<< A list inside a list:

**##describe<<**

# description of lists etc.

,item1.1=lSum.V.n ##<< Description of list item 1.1

,item1.2=Var1.V.n ##<< Description of list item 1.2

**##end<<**

# necessary to end enclosed 'describe'

)

,item2 ##<< Description of item 2

)

lSum.V.n

**##value<<**

## Description of the return/output values of your function (mandatory)

}

## Automated R5 reference class documentation

R5 reference classes are not (yet) implemented in the package **inlinedocs**. Therefore all methods of the sEddyProc class are converted to normal functions in the package generation script (genPackage.R) prior to documentation and reconverted afterwards. The conversion of methods to functions is done by omitting 'sEddyProc$methods(', replacing '<<-' with <- and '})' with '}'. The R5 description is provided in hyphens at the beginning of the function.

sEddyProc$methods(

sFunction = function(

...

)

...

{

'R5 reference class only: Description text for automated R5 documentation'

... <<- ...

})

Attention: Since '})' will be replaced with '}' for the conversion of methods to functions, (three) white spaces need to be added to the bracket combination ('} )') in all other places within the Eddy\*.R scripts!

## Automated example documentation

Examples implemented as attribute to the function will automatically appear in the documentation:

attr(fTemplateInlinedocs,"ex") <- function(){

# Example code demonstrating the functionality of your function

x.V.n <- 1:10

fTemplateInlinedocs(x.V.n,x.V.n\*2)

}

## Script file naming for automated documentation

Attention: The dependencies of the files need to be represented in their alphabetical order! E.g. the file Eddy.R with the R5 class initialization is named so that it is listed before all other files with methods of this class (like REddyGapFilling.R).

## Non-automated documentation

Self-written documentation files can be provided in inst/develop/genDocu and will overwrite the automatically generated documentation in the man directory. The following two files are provided: The package documentation REddyProc-package.Rd and the data documentation Example\_DETha98.Rd.

# References

1 Reichstein, M. *et al.* On the separation of net ecosystem exchange into assimilation and ecosystem respiration: review and improved algorithm. *Global Change Biology* **11**, 1424-1439 (2005).

2 Papale, D. *et al.* Towards a standardized processing of net ecosystem exchange measured with eddy covariance technique: algorithms and uncertainty estimation. *Biogeosciences* **3**, 571-583 (2006).